Querying and Merging Heterogeneous Data by Approximate Joins on Higher-Order Terms

Simon Price and Peter Flach

Department of Computer Science, University of Bristol, Bristol BS8 1UB, United Kingdom.
{simon.price,peter.flach}@bristol.ac.uk

Abstract. Integrating heterogeneous data from sources as diverse as web pages, digital libraries, knowledge bases, the Semantic Web and databases is an open problem. The ultimate aim of our work is to be able to query such heterogeneous data sources as if their data were conveniently held in a single relational database. Pursuant to this aim, we propose a generalisation of joins from the relational database model to enable joins on arbitrarily complex structured data in a higher-order representation. By incorporating kernels and distances for structured data, we further extend this model to support approximate joins of heterogeneous data. We demonstrate the flexibility of our approach in the publications domain by evaluating example approximate queries on the CORA data sets, joining on types ranging from sets of co-authors through to entire publications.

1 Introduction

An increasingly important problem is the integration of data from sources as diverse as web pages, digital libraries, knowledge bases, the Semantic Web and databases that, collectively, are referred to as heterogeneous data. Integration allows an application to query the data using a single query language, just as if the data were a single homogeneous data source.

In this paper we combine two contrasting knowledge representational approaches into a single coherent formalism that is well suited to the integration of heterogeneous data. The first of these representational approaches, the relational model, is widely used as the basis for relational databases and is accompanied by a well-defined algebra for manipulating relational data. However, relational representations of complex structured data can be difficult to design and even more difficult for a human to read. The second representational approach, terms in a higher-order logic, offers a more human-readable representation of structured data than the relational model but has no well-defined analogue of the relational algebra for the querying of its terms. The formalism we introduce here is a subset of relational algebra upgraded for terms in a higher-order logic, bringing the well known and widely used join operator of relational algebra to the knowledge representational formalism of higher-order terms. This new algebra incorporates a generalisation of the relational model to higher-order terms and we show that a join operator from the relational model may be viewed as a special case of the higher-order join.

Data integration typically transforms heterogeneous data formats into a single homogeneous data format, usually into the format which has the most convenient algebra

for data integration rather than the format with the most natural representation of the data. In an ideal data integration scenario, where no uncertainty exists in the correspondences between individuals from different data sources, the homogeneous data format chosen is merely a technical implementation detail and places no restrictions on what may be reliably integrated. Unfortunately, the more diverse the sources of data being integrated, the more likely it will be that the integration involves a degree of uncertainty – for example, in identifying correspondences between individuals from different data sources. In order to automate such integration tasks, approximate matching techniques from statistics, machine learning and data mining may be employed. However, the transformation of data to the most widely used homogeneous format, relational data, obscures the data's natural type and structure, unnecessarily complicating the application of approximate matching techniques. In this paper, we show that approximate matching can take place without this obfuscating transformation to a relational representation. Our approach to approximate matching for data integration uses kernels and distances applied directly to representations of indivuals as (closed) terms in a higher-order logic.

The outline of the paper is as follows: Section 2 reformulates the traditional relational join from the relational model. Section 3 introduces the knowledge representational formalism and describes a family of kernels and distances on that formalism. Section 4 upgrades the relational join to handle structured data. Section 5 investigates the application of joins for structured data. The remaining sections review related work and future directions before concluding.

2 Relational Joins

We first define the relational join in its exact form and then adapt this to define the approximate relational join. Our definitions differ from those of the traditional relational model in that we have not embedded attribute names, which are in fact database-specific schema metadata, into the data representation. We assume throughout that there is an associated schema for each relation. By keeping the metadata separate from the data we achieve a more elegant upgrade from the relational model to the structured data model.

2.1 Exact Relational Joins

An n-tuple $(x_1, \ldots, x_n)$ is an element in $D_1 \times \cdots \times D_n$ where $x_1, \ldots, x_n$ are values drawn from domains $D_1, \ldots, D_n$ respectively. We refer to n-tuples as *tuples* unless the value of n is significant. Item $i \in \{1, \ldots, n\}$ of a tuple $t = (x_1, \ldots, x_n)$ is the value x_i and is written $t|_i$. A *relation* R of degree n is a finite set of n-tuples such that $R \subseteq D_1 \times \cdots \times D_n$ where $D_1, \ldots, D_n$ are domains, which need not necessarily be distinct. The *relation index* I_R of a relation R of degree n is the set $\{1, \ldots, n\}$.

Definition 1 (θ-Restriction). *Let θ be a predicate $\theta : D \times D \to \mathbb{B}$ for some domain D. If A and B are relations with tuple items $a|_i \in D$ and $b|_j \in D$ respectively for some $(i, j) \in I_A \times I_B$, then the* θ-restriction *$\sigma_{i\theta j}$ is defined on $T \subseteq A \times B$ as follows: $\sigma_{i\,\theta j}(T) = \{(a, b) \mid a|_i \,\theta\, b|_j \ \wedge\ (a, b) \in T\}$.*

The infix θ in the subscript of σ follows the historical convention from the relation database literature and so $i\,\theta j$, or equivalently $\theta(i,j)$, does not mean that θ applies to i and j; instead $i\,\theta j$ is shorthand notation for the membership test $a|_i\,\theta\ b|_j$ for all $(a,b) \in T$. The operator θ is typically drawn from the set $\{=, \neq, <, \leq, >, \geq\}$ but does not necessarily have to come from this set. θ-restriction is often just referred to as *restriction* and in such cases θ is assumed to be the equality operator. The name *selection* is often used instead of *restriction*, but the latter avoids confusion with the similarly named and better known `select` operator from SQL which has a somewhat different meaning. Restriction is also sometimes defined as *generalised restriction*: Let φ be a proposition that consists of atoms as allowed in θ-restriction and the logical operators $\wedge$, $\vee$ and $\neg$, then if A and B are relations, the generalised restriction σ_φ is defined on $T \subseteq A \times B$ as $\sigma_\varphi(T) = \{\ t \mid \varphi(t)\ \wedge\ t \in T\}$. Standard results show that generalised restrictions can always be expressed as combinations of θ-restrictions. θ-restriction is thus the basis of the following fundamental relational join operator.

Definition 2 (θ-Join). *Let θ be a predicate $\theta : D \times D \to \mathbb{B}$ for some domain D. If A and B are relations with tuple items $a|_i \in D$ and $b|_j \in D$ respectively for some $(i,j) \in I_A \times I_B$, then the* θ-join $\bowtie_{i\,\theta j}$ *of A and B is defined as $A \bowtie_{i\,\theta j} B = \sigma_{i\,\theta j}(A \times B)$.*

When θ is equality the θ-join is called the *equi-join*. By replacing the θ-restriction operator in the θ-join by the generalised restriction operator we arrive at the definition of the *generalised join*: Let φ be a proposition that consists of atoms as allowed in θ-restriction and the logical operators $\wedge$, $\vee$ and $\neg$. If A and B are relations then the *relational join* $\bowtie_\varphi$ is defined as $A \bowtie_\varphi B = \sigma_\varphi(A \times B)$. Other non-fundamental joins include the *natural join*, *semijoin*, *antijoin*, *outer joins* and *inner joins* [1, 2]. For the purposes of upgrading relational joins to handle structured data, it is sufficient to consider just the θ-join and optionally, as a useful syntactic convenience, the generalised join.

2.2 Approximate Relational Joins

In order to turn an exact relational join into an approximate one it is necessary to replace the exact θ operator in θ-restriction with a suitable approximate version. For example, substituting exact equality $=$ with an approximate equality $\approx$ enables joining on tuple items that are either the same or in some way sufficiently similar.

One method of implementing approximate equality $\approx$ is to use a distance metric or pseudo-metric *dist*, defined on the domain of a pair of relational tuple items, together with a threshold δ to define a *proximity relation*.

Definition 3 (Proximity). *If the function $dist : D \times D \to \mathbb{R}$ is a distance on pairs of values from some domain D and $\delta \in \mathbb{R}$ $(\delta \geq 0)$ is a threshold then* proximity *is a predicate $\approx : D \times D \to \mathbb{B}$ defined by*

$$\forall x, y \in D\ \ (x \approx y) \iff \{(x,y) \mid dist(x,y) \leq \delta\ \wedge\ x, y \in D\}.$$

By the definition of distance, the co-domain of *dist* is not constrained to have an upper bound. Some normalising function φ may be used to apply an upper bound to a distance. The function $\varphi : \mathbb{R} \to \mathbb{R}$ must be a non-decreasing function from the positive

reals into some closed interval, typically $[0, 1]$, such that $\varphi(0) = 0$, $\varphi(v) > 0$ if $v > 0$, and $\varphi(v + u) \leq \varphi(v) + \varphi(u)$, for each v and y. Example choices of φ from [3] are $\varphi(v) = \mathit{min}(v, 1)$ or $\varphi(v) = \frac{v}{v+1}$. Alternatively, the normalisation may be performed in the feature space of $x, y \in D$ so that $\mathit{dist}(x, y)$ is inherently normalised. For example, if the distance is derived from a kernel then a normalising kernel may be used [4].

Definition 4 (Proximity-Join). *Let $\approx : D \times D \rightarrow \mathbb{B}$ be a proximity for some domain D. If A and B are relations with tuple items $a|_i \in D$ and $b|_j \in D$ respectively for some $(i, j) \in I_A \times I_B$, the* proximity-join $\widetilde{\bowtie}_{i \approx j}$ *of A and B is $A \widetilde{\bowtie}_{i \approx j} B = \sigma_{i \approx j}(A \times B)$.*

The same historical notational convention is followed here for the subscripted $\approx$ as for the subscripted θ described earlier for the exact θ-join. The proximity-join as defined here is an approximate analogue of the exact relational equi-join. By choosing other proximity relations that are approximate analogues of exact relations, for example where $\theta \in \{=, \neq, <, \leq, >, \geq\}$, an approximate version of the relational θ-join might be defined. In this paper we restrict our attention to the proximity-join.

3 Representing Structured Data

The relational model is now the de facto standard for database-driven applications in data mining and computing in general, but it is not ideally suited for representing semi-structured data such as Web pages and XML, nor structured data such as the Semantic Web and RDF. However, the representation of such structures in relational databases is commonplace using a multitude of (often tortuous) representations and querying patterns. By contrast, the *individuals-as-terms* model offers straight forward representations of both structured and semi-structured data while at the same time having the representational capacity to represent relations from the relational model. The individuals-as-terms representation is a generalisation of the relational model's attribute-value representation and collects all information about an individual in a single term.

We are not advocating the individuals-as-terms repesentation as a replacement for the general purpose relational representation. But in the context of querying and merging heterogeneous data, the individuals-as-terms representation more transparently models the structure of the data in a way that is both human-readable and that explicitly exposes that structure to machine learning and data mining algorithms.

The knowledge representational formalism we use as our individuals-as-terms representation is *basic terms*, a family of typed terms in higher-order logic, which is based on Church's simple theory of types [5] with several extensions [3]. This formalism has been chosen over the possible alternative of first-order logic because terms in the higher-order logic natively support a variety of data types that are important for representing individuals, including sets, multisets and graphs. Being a strongly typed logic, helps to reduce search spaces and the type of terms provides useful metadata. The theory behind the logic and the individuals-as-terms formalism is set out in [3] and we give only a brief overview here.

We assume an *alphabet* consisting of: $\mathfrak{T}$ the set of type constructors of various arities, $\mathfrak{P}$ the set of parameters, $\mathfrak{C}$ the set of constants, and $\mathfrak{V}$ the set of variables. Included in $\mathfrak{T}$ is the constructor Ω of arity 0 with a corresponding domain of $\{\mathit{True}, \mathit{False}\}$, the

booleans. Types are constructed from type constructors in $\mathfrak{T}$ and type variables in $\mathfrak{P}$ using the symbols $\rightarrow$ for function types and $\times$ for product types. A *type* is defined inductively as follows: (1) Each parameter in $\mathfrak{P}$ is a type. (2) If T is a type constructor in $\mathfrak{T}$ of arity k and $\alpha_1, \ldots, \alpha_k$ are types, then $T\ \alpha_1 \ldots \alpha_k$ is a type. (For $k = 0$, this reduces to a type constructor of arity 0 being a type). (3) If α and β are types, then $\alpha \rightarrow \beta$ is a type. (4) If $\alpha_1, \ldots, \alpha_n$ are types, then $\alpha_1 \times \cdots \times \alpha_n$ is a type. (For $n = 0$, this reduces to 1 being a type). A type is *closed* if it contains no parameters. $\mathfrak{S}^C$ denotes the set of all closed types obtained from an alphabet.

The set of constants $\mathfrak{C}$ includes $\top$ (true) and $\bot$ (false). A *signature* is the declared type for a constant. A constant C with signature α is often denoted $C\ :\ \alpha$. Let $[]$ be the empty list constructor with signature *List* a where a is a parameter and *List* is a type constructor. Let $\#$ be the list constructor with signature $a \rightarrow$ *List* $a \rightarrow$ *List* a.

The *terms* of the logic are the terms of typed λ-calculus and are formed in the usual way by abstraction, tupling and application from constants in $\mathfrak{C}$ and a set of variables. The set of all terms obtained from a particular alphabet is denoted $\mathfrak{L}$. A basic term is the canonical representative of an equivalence class of terms [4, 3]. The set of *basic terms*, $\mathfrak{B}$, is defined inductively as follows: (1) *Basic structures* – If C is a data constructor having signature $\sigma_1 \rightarrow \cdots \rightarrow \sigma_n \rightarrow (T\ a_i \ldots a_k)$, $t_1, \ldots, t_n \in \mathfrak{B}$ $(n \geq 0)$, and t is $C\ t_1 \ldots t_n \in \mathfrak{L}$, then $t \in \mathfrak{B}$. (2) *Basic abstractions* – If $t_1, \ldots, t_n \in \mathfrak{B}$, $s_1, \ldots, s_n \in \mathfrak{B}$ $(n \geq 0)$, $s_0 \in \mathfrak{D}$ and t is λx if $x = t_1$ then s_1 else $\ldots$ if $x = t_n$ then s_n else $s_0 \in \mathfrak{L}$, then $t \in \mathfrak{B}$. (3) *Basic tuples* – If $t_1, \ldots, t_n \in \mathfrak{B}$ $(n \geq 0)$ and t is $(t_1, \ldots, t_n) \in \mathfrak{L}$, then $t \in \mathfrak{B}$. See section 4 for examples and diagrams of basic terms.

3.1 Kernels and Distances for Basic Terms

Kernel functions [6] are an effective way of inducing distances on a wide variety of data structures. One promising recent kernel function for structured data is the default kernel for basic terms introduced in [4].

Definition 5 (Default Kernel for Basic Terms [4]). *The function $k : \mathfrak{B} \times \mathfrak{B} \rightarrow \mathbb{R}$ is defined inductively on the structure of terms in $\mathfrak{B}$ as follows.*

1. *If $s, t \in \mathfrak{B}_\alpha$, where $\alpha = T\alpha_1 \ldots \alpha_k$, for some $T, \alpha_1, \ldots, \alpha_k$, then*

$$k(s,t) = \begin{cases} \kappa_T(C,D) & \text{if } C \neq D \\ \kappa_T(C,C) + \sum_{i=1}^{n} k(s_i,t_i) & \text{otherwise} \end{cases}$$

   where s is $C\ s_1 \ldots s_n$ and t is $D\ t_1 \ldots t_m$.

2. *If $s, t \in \mathfrak{B}_\alpha$, where $\alpha = \beta \rightarrow \gamma$, for some β, γ, then*

$$k(s,t) = \sum_{\substack{u \in supp(s) \\ v \in supp(t)}} k(V(s\ u), V(t\ v)) \cdot k(u,v).$$

3. *If $s, t \in \mathfrak{B}_\alpha$, where $\alpha = \alpha_1 \times \cdots \times \alpha_n$, for some $\alpha_1, \ldots, \alpha_n$, then*

$$k(s,t) = \sum_{i=1}^{n} k(s_i,t_i),$$

where s is $(s_1, \ldots, s_n)$ and t is $(t_1, \ldots, t_n)$.

4. *If there does not exist $\alpha \in \mathfrak{S}^c$ such that $s, t \in \mathfrak{B}_\alpha$, then $k(s,t) = 0$.*

The definition, assumes that for each type constructor $T \in \mathfrak{T}$, $\kappa_T : \mathcal{X}_T \times \mathcal{X}_T \to \mathbb{R}$ is a kernel on the set of data constructors $\mathcal{X}_T$ associated with T. Below we give an example of the calculation of the default kernel for an example data structure: sets of strings.

Example 1 (Default Kernel on Sets of Strings). Let S be a nullary type constructor for strings and $A, B, C, D : S$. Choose κ_S and κ_Ω to be the matching kernel. Let s be the set $\{A, B, C\} \in \mathfrak{B}_{S \to \Omega}$, $t = \{A, D\}$, and $u = \{B, C\}$. Then

$$\begin{aligned}
k(s,t) &= k(A,A)k(\top,\top) + k(A,D)k(\top,\top) + k(B,A)k(\top,\top) \\
&\quad + k(B,D)k(\top,\top) + k(C,A)k(\top,\top) + k(C,D)k(\top,\top) \\
&= \kappa_S(A,A)\kappa_\Omega(\top,\top) + \kappa_S(A,D)\kappa_\Omega(\top,\top) + \kappa_S(B,A)\kappa_\Omega(\top,\top) \\
&\quad + \kappa_S(B,D)\kappa_\Omega(\top,\top) + \kappa_S(C,A)\kappa_\Omega(\top,\top) + \kappa_S(C,D)\kappa_\Omega(\top,\top) \\
&= \kappa_S(A,A) + \kappa_S(A,D) + \kappa_S(B,A) + \kappa_S(B,D) \\
&\quad + \kappa_S(C,A) + \kappa_S(C,D) \\
&= 1 + 0 + 0 + 0 + 0 + 0 \\
&= 1.
\end{aligned}$$

Similarly, $k(s,u) = 2$ and $k(t,u) = 0$.

Noting that valid positive semi-definite kernels induce pseudo-metrics [4], this allows the derivation of a distance from any such kernel, including the kernel for basic terms, as follows. Let $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a kernel on $\mathcal{X}$. The distance measure induced by k is defined as $d_k(s,t) = \sqrt{k(s,s) - 2k(s,t) + k(t,t)}$. If k is a valid kernel the d_k is well behaved in that it satisfies the conditions of a pseudo-metric. Continuing the earlier sets of strings example, the following example illustrates the calculation of a distance from the default kernel for basic terms.

Example 2 (Default Distance on Sets of Strings). Let $s = \{A, B, C\}$, $t = \{A, D\}$, and $u = \{B, C\}$ where $s, t, u \in \mathfrak{B}_{S \to \Omega}$. We have $k(s,s) = 3$, $k(t,t) = 2$ and $k(u,u) = 2$. Then, $d_k(s,t) = \sqrt{3 - 3 + 2} = 1.73$, $d_k(s,u) = \sqrt{3 - 4 + 2} = 1$, and $d_k(t,u) = \sqrt{2 - 0 + 2} = 2$.

However, one of the strengths of the default kernel is that it allows any other valid kernel to be associated with a specific type. For example, the following *p*-spectrum kernel, defined on strings, is used in our experiments later in the paper.

Definition 6 (*p*-Spectrum Kernel [6]). *The feature space F associated with the* p*-spectrum kernel is indexed by $I = \Sigma^p$, with the explicit embedding from the space of all finite sequences over and alphabet Σ to a vector space F and is given by $\phi^p_u(s) = |\{(v_1, v_2) : s = v_1 u v_2\}|, u \in \Sigma^p$. The associated kernel is defined as $\kappa_p(s,t) = \langle \phi^p(s), \phi^p(t) \rangle = \sum_{u \in \Sigma^p} \phi^p_u(s)\phi^p_u(t)$.*

4 Relational Joins for Structured Data

We now upgrade both exact and approximate relational joins for structured data. The way we achieve this is to first upgrade the knowledge representation of the relation to be a set of basic terms rather than the traditional set of tuples. We then upgrade the relation index so that it indexes parts of a basic term rather than the traditional parts of a tuple. Once these two steps are completed, upgrading the exact relational join follows almost automatically with only modest changes to the definitions of the θ-restriction and joins. The final step then brings together the default kernel for basic terms and the approximate join to arrive at the main result of an approximate relational join for structured data. So to begin, we first upgrade the relation from section 2.1 to become the *basic term relation*, which is a basic term of type $\alpha \rightarrow \Omega$.

Definition 7 (Basic Term Relation). *A* basic term relation $R \subseteq \mathfrak{B}_\alpha$ *is a finite set of basic terms of the same type.*

In order to upgrade the relation index from section 2.1 so that it is applicable to the basic term relation, a suitable method of indexing sub-parts of a basic term is required. Recall that well-formed basic terms can consist of basic structures (e.g. lists, trees), basic abstractions (e.g. sets, multisets), basic tuples or arbitrary combinations of these three.

4.1 Indexing Basic Terms

In the logic, sub-parts of a term are referred to as subterms and so we are concerned with indexing the subterms of a basic term. The standard method for indexing subterms in the logic enumerates a decomposition of a given term such that every subterm is labelled with a unique string [3].

However, we introduce an alternative approach to indexing that, instead of enumerating all subterms of a term, defines a *type tree index set* over all subtypes of the type of a basic term. To do this we first adopt the definition of a type tree from [7] and then define a different annotation of the tree such that every member of the type tree index set identifies a set of terms rather than a single term. This ensures any index defined on a type is meaningful across all terms of that type. Furthermore, the set of subterms identified is guaranteed to consist entirely of well-formed basic terms.

To achieve this we follow the same interpretation of subtypes as [3] and restrict our attention to basic terms whose basic structures are in canonical form as defined below.

Definition 8 (Basic Structures in Canonical Form). *A type* $\tau = T\ \alpha_1 \ldots \alpha_k$ *is a basic structure in canonical form when, for all data constructors* $C_i : \tau_{i1} \rightarrow \cdots \rightarrow \tau_{in} \rightarrow \tau$ *that are associated with* T*, all the types of* τ_{ij} *are subtypes of* τ*.*

We begin our definition of the type tree index set with some preparatory notation. Let $\mathbb{Z}^+$ denote the set of positive integers and $(\mathbb{Z}^+)^*$ the set of all strings over the alphabet of positive integers, with ε denoting the empty string. io denotes the string concatenation of i with o where $i \in \mathbb{Z}^+$ and $o \in (\mathbb{Z}^+)^*$.

Definition 9 (Type Tree Index Set). *The* type tree index set *of a canonical type τ, denoted $\mathcal{O}(\tau)$, is the set of strings in $(\mathbb{Z}^+)^*$ defined inductively on the structure of τ.*

1. *If τ is an atomic type, then $\mathcal{O}(\tau) = \{\varepsilon\}$.*
2. *If τ is a basic structure type $\tau = T\ \alpha_1 \ldots \alpha_n$ in canonical form, with data constructors $C_i : \tau_{i_1} \to \cdots \to \tau_{i_m} \to \tau$ for all $i \in \{1, \ldots, l\}$, then $\mathcal{O}(\tau) = \{\varepsilon\} \cup \bigcup_{v=1}^{p} \{\, vo_v \mid o_v \in \mathcal{O}(\xi_v)\}$, where $\xi_1, \ldots, \xi_p$ are the types from α_k where $\alpha_k = \tau_{i_j}$ and $\tau_{i_j} \neq \tau$, and assuming that for every $\tau_{i_j} \neq \tau$ there exists an α_k such that $\alpha_k = \tau_{i_j}$.*
3. *If τ is a basic abstraction type $\beta \to \gamma$, then $\mathcal{O}(\tau) = \{\varepsilon\} \cup \{\, 1o \mid o \in \mathcal{O}(\beta)\} \cup \{\, 2o \mid o \in \mathcal{O}(\gamma)\}$.*
4. *If τ is a basic tuple type $\tau = \tau_1 \times \cdots \times \tau_n$, then $\mathcal{O}(\tau) = \{\varepsilon\} \cup \bigcup_{i=1}^{n} \{\, io_i \mid o_i \in \mathcal{O}(\tau_i)\}$.*

Part 1, the base case, states that types for which all the associated data structures have arity zero, such as Ω (the type of the booleans), *Int* (the type of the integers), and *Char* (the type of characters), have a singleton index set containing the empty string. Part 2 states that each subtype that occurs in the signatures of the associated data constructors, and that is not itself of type τ of the basic structure, is labelled with a unique string. Part 3 labels the β and γ types of basic abstractions with a pair of unique strings. Similarly, part 4 labels each tuple item in a basic tuple with a unique string.

The significance of defining indexing on the type tree of basic terms rather than on the terms themselves is that each member of a type tree index set $o \in \mathcal{O}(\tau)$ is not uniquely tied to any individual term of type τ. This increases the generality of the indexing such that each member of the type tree index set for type τ identifies, for any basic term $t : \tau$, an equivalence class of subterms rather than a single term. Thus $\mathcal{O}(t)$ induces a set of equivalence classes on the subterms of t. We refer to the set of subterms identified with a given member (index) of the type tree index set as the *basic subterm set* at that index.

Definition 10 (Basic Subterm Set). *If t is a basic term of type τ and $o \in \mathcal{O}(\tau)$ then the* basic subterm set of t at type tree index *o, denoted $t|_o$, is defined inductively on the length of o as follows.*

1. *If $o = \varepsilon$, then $t|_o = \{t\}$.*
2. *If $o = jo'$, for some o', and t has the form $C\ t_1 \ldots t_m$, with associated type $T\ \alpha_1 \ldots \alpha_n$, then $t|_o = s_j|_{o'}$ where $s_j = t_i : \tau_i$ such that $\tau_i \neq \tau$ and $\tau_i = \alpha_j$.*
3. *If $o = 1o'$, for some o', and t has the form* if_then_else(u, v, s)*, then $t|_o = u|_{o'} \cup s|_o$.*
4. *If $o = 2o'$, for some o', and t has the form* if_then_else(u, v, s)*, then $t|_o = v|_{o'} \cup s|_o$.*
5. *If $o = io'$, for some o', and t has the form $(t_1, \ldots, t_n)$, then $t|_o = t_i|_{o'}$, for $i = 1, \ldots, n$.*

A basic subterm set *is a set of basic subterms of a basic term at some type tree index. A basic subterm is* proper *if it is not at type tree index ε.*

Basic subterms indexed in part 1, the base case, are singleton sets containing an atomic term. Basic subterms indexed in part 2 are basic structures. Basic subterms indexed in parts 3 and 4 are the support and value of basic abstractions, i.e. respective instances of α and β, from $\alpha \to \beta$. Basic subterms indexed in part 5 are basic tuples.

Below we give examples of a type tree index set and basic subterm sets for each of basic tuples, basic structures, and basic abstractions. Starting with basic tuples in Example 3 where it can be seen that type-based indexing identifies all the tuple items, but as singleton sets, and in addition it identifies the reflexive term at $t|_\varepsilon$.

Example 3. If basic tuple $t \in \mathfrak{B}_{M\times N\times O\times P}$ is the term $t = (A, B, C, D)$, where $A : M$, $B : N$, $C : O$, $D : P$, then the type tree index set of t is $\mathcal{O}(t) = \{\varepsilon, 1, 2, 3, 4\}$, the derivation of which can be seen from Fig.1. The basic subterm sets of t are $t|_\varepsilon = \{(A, B, C, D)\}$, $t|_1 = \{A\}$, $t|_2 = \{B\}$, $t|_3 = \{C\}$ and $t|_4 = \{D\}$.

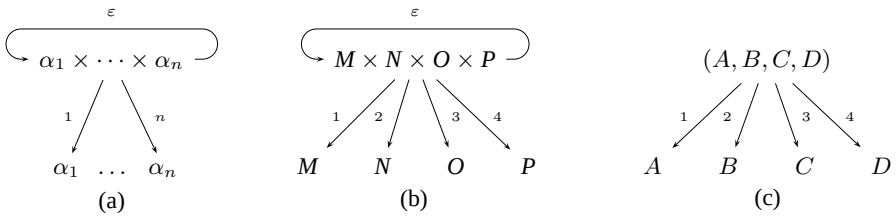

Fig. 1: Type-based indexing for basic tuples. (a) Type tree index for n-tuples of type $\alpha_1 \times \cdots \times \alpha_n$. (b) Type tree index for 4-tuples of type $M \times N \times O \times P$. (c) Basic subterm tree for term (A, B, C, D) where $A : M$, $B : N$, $C : O$, $D : P$.

Representing basic structures, the usual right branching representation of lists is given in Example 4, where the basic subterm set at $t|_1$ captures one meaning of a list as a set of values and $t|_\varepsilon$ captures the meaning of a list as a set of sequences.

Example 4. If τ is a type of lists such that $\tau = List\ M$, where $M \subseteq \mathfrak{B}$ is a nullary type constructor, with associated data constructors $\#$ and $[]$, having signatures $[] : List\ M$, and $\# : M \to List\ M \to List\ M$, then the type tree index set of τ is $\mathcal{O}(\tau) = \{\varepsilon, 1\}$. If basic terms $s, t \in \mathfrak{B}_{List\ M}$ are the lists $s = [A, B, C]$ and $t = [A, D]$, then as can be seen from Fig.2, the basic subterm sets of s and t are $s|_\varepsilon = \{[A, B, C], [B, C], [C], []\}$, $s|_1 = \{A, B, C\}$, and $t|_\varepsilon = \{[A, D], [D], []\}$, $t|_1 = \{A, D\}$.

For basic abstractions, a set is given in Example 5 and a multiset in Example 6. For both sets and multisets, $t|_1$ captures the meaning as a set of values whereas $t|_2$ will always be $\{\top\}$ for sets and a set of multiplicities for multisets. A corollary of Definition 9 is that the type tree index set of a basic abstraction type is always $\{\varepsilon, 1, 2\}$.

Example 5. If τ is a basic abstraction type representing sets such that $\tau = M \to \Omega$, where $M \subseteq \mathfrak{B}$ is a nullary type constructor, then the type tree index set of τ is $\mathcal{O}(\tau) = \{\varepsilon, 1, 2\}$. If basic term $t = \{A, B, C\}$, where $A, B, C : M$, then the basic subterm sets are $t|_\varepsilon = \{\{A, B, C\}\}$, $t|_1 = \{A, B, C\}$ and $t|_2 = \{\top\}$.

Example 6. If τ is a basic abstraction type representing multisets such that $\tau = M \to Nat$, where $M \subseteq \mathfrak{B}$ is a nullary type constructor and Nat, then the type tree index set of τ is $\mathcal{O}(\tau) = \{\varepsilon, 1, 2\}$. If basic term $t = \{A, A, A, B, C, C\}$, where $A, B, C : M$, then the basic subterm sets are $t|_\varepsilon = \{\{A, A, A, B, C, C\}\}$, $t|_1 = \{A, B, C\}$ and $t|_2 = \{1, 2, 3\}$.

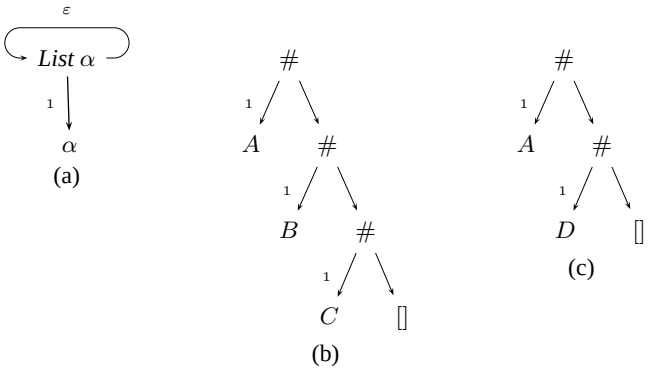

Fig. 2: Type-based indexing for basic structures. (a) Type tree index for *List* α. (b) and (c) Basic subterm trees for terms $[A, B, C]$ and $[A, D]$ of type *List M* where $A, B, C, D : M$.

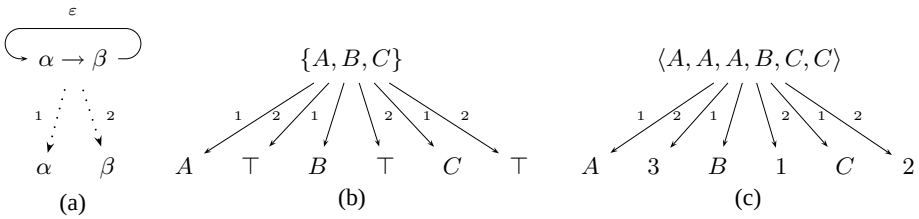

Fig. 3: Type-based indexing for basic abstractions. (a) Type tree index for type $\alpha \to \beta$. (b) Basic subterm tree for set $\{A, B, C\}$, type $M \to \Omega$, where $A, B, C : M$ and $\top : \Omega$. (c) Basic subterm tree for multiset $\langle A, A, A, B, C, C\rangle$, type $M \to$ Nat, where $A, B, C : M$ and $1, 2, 3 :$ Nat.

A useful and straight forward reformulation of type-based indexing is *type name-based indexing* that, instead of enumerating the edges of the type tree, directly labels the vertices of the type tree. The simplest approach being to assign a unique type name to every vertex in the type tree. If the names assigned have no understandable meaning to humans then this method offers no advantages over type-based indexing. However, if the knowledge representational formalism used to define types and data instances uses human-understandable names then type name-based indexing provides a useful notation for referring to basic subterm sets, as illustrated in Example 7.

Example 7. Let *Author* be the type of authors from the publications domain, which define declaratively in the Haskell style syntax from [4] as follows.

```
type Author = (Name,Publications);
type Name = String;
type Publications = List Publication;
type Publication = (Mode,Coauthors,Title,Venue,Year);
data Mode = Journal | Proceedings | ... | Book;
type Coauthors = Coauthor -> Bool;
type Coauthor = String;
type Title = String;
type Venue = String;
type Year = Int;
```

This states that *Author* is a pair of *Name* and *Publications*, where *Name* is an alias for *String* the type of strings, and *Publications* is a list of publications, which in turn is a 5-tuple of *Mode*, *Coauthors*, . . . , *Year*, where *Mode* has the nullary data constructors *Journal*, *Proceedings*, . . . , *Book*, and so on through to *Year* which is an alias for the type *Int*, the type of the integers. *Coauthors* is a basic abstraction from *Coauthor* to *Bool*, where *Bool* is the type Ω, i.e. *Coauthors* is a set of *coauthors*. To ensure the required uniqueness of type names *Coauthor*, *Title*, *Venue* and *Name* are aliases for the type *String*. The type tree index set is thus $\{$*Author, Author.Name, Author.Publications.Publication.Mode*, . . . , *Author.Publications.Publication.Year*$\}$.

A type tree index set generated using this method is isomorphic with that produced by Definition 9, as illustrated informally in Fig.4. The constraint that all basic subtypes must be uniquely named permits the following simpler definition of a basic subterm set.

Definition 11 (Basic Subterm Set (with named types)). *If t is a closed basic term of type τ and $\alpha \subseteq \tau$ then the* basic subterm set of t at type α, *denoted $t|_\alpha$, is $t|_\alpha = \{s \mid s$ occurs in t with type $\alpha\}$. A* basic subterm set *is a set of basic subterms of a basic term at some type tree $\alpha \subseteq \mathfrak{B}$. A basic subterm is* proper *if $\alpha \neq \tau$.*

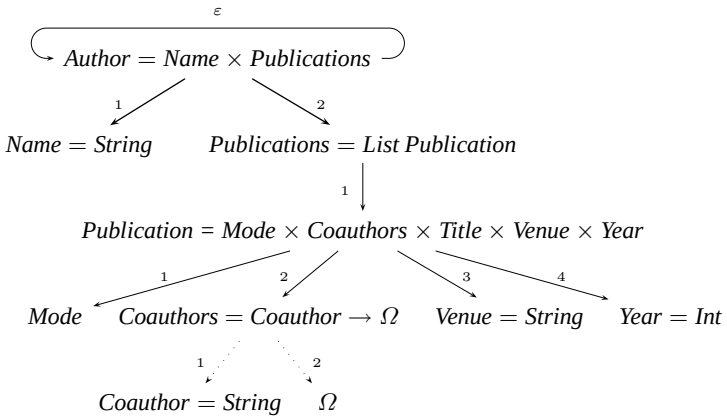

Fig. 4: Type name-based and type-based indexing for type *Author*.

4.2 Indexing Basic Term Relations

Having upgraded our representation of a relation $R : \tau$ to handle structured data represented as basic terms, and having chosen a suitable indexing method for the basic subterm set $\mathcal{O}(\tau)$, we are now able to conveniently define the *basic term relation index* as the structured data counterpart of the relation index.

Definition 12 (Basic Term Relation Index). *The* basic term relation index I_R *of a basic term relation R of type τ is $I_R = \mathcal{O}(\tau)$.*

4.3 Exact Relational Join for Structured Data

The upgraded definitions of an exact relation join for structured data closely follow the earlier relational definitions but now using basic term relations and indexing.

Definition 13 (Basic Term Projection). *If $t \in \tau$, where $\tau \subseteq \mathfrak{B}$, then the* basic term projection *π of t on $i \in I_t$ is $\pi_i(t) = \{s \mid s$ is the basic subterm at type tree index $i\}$.*

A basic term projection $\pi_i(t)$ may also be written as $t|_i$.

Definition 14 (Basic Term Generalised Projection). *If $t \in \mathfrak{B}$ then the* basic term generalised projection *π of t on $\rho \subseteq I_t$ is the set $\pi_\rho(t) = \{\ t|_i \ \mid \ i \in \rho\ \}$.*

Definition 15 (Basic Term θ-Restriction). *Let θ be a predicate $\theta : (\mathfrak{B}_\alpha \to \Omega) \to (\mathfrak{B}_\alpha \to \Omega) \to \Omega$ for some $\alpha \in \mathfrak{B}_\alpha$. If A and B are basic term relations with basic terms $a|_i \subseteq \mathfrak{B}_\alpha$ and $b|_j \subseteq \mathfrak{B}_\alpha$ respectively for some $(i,j) \in I_A \times I_B$, then* basic term θ-restriction *$\sigma_{i\theta j}$ is defined on $T \subseteq A \times B$ as $\sigma_{i\theta j}(T) = \{(a,b) \mid\ a|_i\, \theta\, b|_j \wedge (a,b) \in T\}$.*

The predicate $\theta : (\mathfrak{B}_\alpha \to \Omega) \to (\mathfrak{B}_\alpha \to \Omega) \to \Omega$ is defined on sets of basic terms. In other words, θ is a predicate on basic term relations.

Definition 16 (Basic Term Generalised Restriction). *Let φ be a proposition that consists of atoms as allowed in basic term θ-restriction and the logical operators $\wedge$, $\vee$ and $\neg$. If A and B are basic term relations then the* basic term generalised restriction *σ_φ is defined on $T \subseteq A \times B$ as $\sigma_\varphi(T) = \{\ t \mid \varphi(t)\ \wedge\ t \in T\}$.*

Definition 17 (Basic Term θ-Join). *Let $\theta : (\mathfrak{B}_\alpha \to \Omega) \to (\mathfrak{B}_\alpha \to \Omega) \to \Omega$ be a predicate for some type $\alpha \in \mathfrak{B}$. If A and B are basic term relations with basic terms $a|_i \subseteq \mathfrak{B}_\alpha$ and $b|_j \subseteq \mathfrak{B}_\alpha$ respectively for some $(i,j) \in I_A \times I_B$ then the* basic term θ-join *$\bowtie_{i\,\theta j}$ of A and B is defined as $A \bowtie_{i\,\theta j} B = \sigma_{i\,\theta j}(A \times B)$.*

Definition 18 (Basic Term Generalised Join). *Let φ be a proposition that consists of atoms as allowed in basic term θ-restriction and the logical operators $\wedge$, $\vee$ and $\neg$. If A and B are basic term relations then the* basic term join *$A \bowtie_\varphi B = \sigma_\varphi(A \times B)$.*

4.4 Approximate Relational Join for Structured Data

We assume some distance for basic terms and note that positive semi-definite kernels induce pseudo-metric distances. One suitable kernel is the kernel for basic terms from [4] and described earlier, but other kernels and distances may also be suitable.

Definition 19 (Basic Term Proximity-Join). *Let $\approx : \mathfrak{B}_\alpha \times \mathfrak{B}_\alpha \to \Omega$ be a proximity for some $\mathfrak{B}_\alpha$ of type α. If A and B are basic term relations with subterms $a|_i \in \mathfrak{B}_\alpha$ and $b|_j \in \mathfrak{B}_\alpha$ respectively for some $(i,j) \in I_A \times I_B$, then the* proximity-join *$\widetilde{\bowtie}_{i\approx j}$ of A and B is defined as $A \,\widetilde{\bowtie}_{i\approx j}\, B = \sigma_{i\approx j}(A \times B)$.*

This definition closely parallels that of the approximate relational join on account of the following: the basic term relation is a set which allows the same set theoretic operators from the relational case to apply; the basic term relation index fulfills the same role as the relation index from the relational case; and, finally, the kernel for basic terms' own inductive definition implicitly handles the often recursive nature of structured data. The closeness in form of the definition of the basic term join to that of the relational join facilitates the following result.

Proposition 1. *Relational joins are a special case of basic term relational joins.*

Proof. Assume relation $R \subseteq D_1 \times \cdots \times D_n$ for some domains $D_1, \ldots, D_n$. Assume appropriate type constructors and data constructors such that $D_1, \ldots, D_n \subseteq \mathfrak{B}$. Let basic term relation $S \subseteq D_1 \times \cdots \times D_n$. Let I_R be the relation index of R and I_S be the basic term relation index of S. Clearly there is a surjection from I_R into I_S and thus from the set of tuple items in each tuple in R to the set of subterms in each corresponding basic term tuple in S. Assume the θ operators are available for basic terms and the result follows. □

5 Applications

We have implemented the higher-order relational projection, restriction and join operators and a range of supporting kernels, including the kernel for basic terms, in Prolog. Although Prolog does not natively support the higher-order logic necessary to represent data as basic terms, emulation of typed data, basic tuples, basic structures and basic abstractions (including sets and multisets) has proven to be unproblematic in practice. We are currently working to characterise and evaluate this framework using the application domain of bibligraphic publications. Heterogenous data sets within this domain include CORA, DBLP, Citeseer and Google Scholar. Interesting higher-order approximate joins between pairs (A, B) of these datasets might, for instance, include the following.

- $A \mathbin{\widetilde{\bowtie}}_{\textit{Author.name}} B$, authors in A and B that have similarly names
- $A \mathbin{\widetilde{\bowtie}}_{\textit{Author.affiliation}} B$, authors in A and B affiliated to the same institution
- $A \mathbin{\widetilde{\bowtie}}_{\textit{Author}} B$, authors in A and B similar across all their properties
- $A \mathbin{\widetilde{\bowtie}}_{\textit{Publication.venue}} B$, publications in A and B from the same venue
- $A \mathbin{\widetilde{\bowtie}}_{\textit{Publication.coauthors}} B$, publications in A and B with similar coauthors

For the sake of evaluation we require the ground truth V for each join to be evaluated, where $V \subseteq A \mathbin{\widetilde{\bowtie}} B$ and, for the case where the individuals as terms represent publications, $V = \{(a, b) \mid a \in A \wedge b \in B \wedge a \textit{ and } b \textit{ are variants of the same publication}\}$. The goal is to reconstruct V as $V' = A \mathbin{\widetilde{\bowtie}_s} B$ by choosing an appropriate s from the intersection of the basic subterm sets of A and B. In reality, V is not usually available for pairs of different data sets. For this reason we narrow down our initial evaluation to consider self-joins, $A \mathbin{\widetilde{\bowtie}_s} A$, on a single data set $A = \textit{CORA}$, for which ground truths

are available [8]. *CORA* consists of bibliographic citations, hand-labelled with unique identifiers so that variant citations of the same paper share the same identifier[1].

Given this supervised learning setting, a number of distance-based methods could be used to implement the approximate join, including k-means, k-NN, and agglomerative hierarchical clustering. We chose the latter for this initial investigation on the basis that it produces a dendrogram that is useful in visualising and charactering the join. Although this is a clustering method more normally associated with unsupervised learning, here we are able to make use of the ground truth labelling to achieve a supervised setting. A dendrogram represents a progressive series of joins (or clusterings), with instances in the same cluster being leaves of the same sub-tree. The distance value at each non-terminal node represents a potential threshold δ at which to 'cut' the tree and arrive at set of clusters. δ is the threshold from the proximity predicate from Definition 19. To evaluate the quality of the clustering at a given δ we consider whether each pair of instance data is correctly classified as being in the same cluster or in different clusters; in other words we evaluate a binary classification on all pairs of instances to determine if the two instances in a pair refer to the same publication or to different publications. A confusion matrix is then calculated in order to determine precision and recall for this specific value of δ. To characterise a proximity-join across a range of thresholds we vary δ across the length of the tree and plot a precision-recall chart.

To represent the publications we choose the following type structure[2].

```
type Publications = Publication -> Bool;
type Publication = (Coauthors,Title,Venue,Year);
type Coauthors = Coauthor -> Bool;
type Coauthor = String;
type Title = String;
type Venue = String;
type Year = String;
```

Hence by representing CORA as a basic terms relation of type *Publications*, where $\mathfrak{B}_{Publications} \in \mathfrak{B}$, we are able to execute the following basic term proximity-joins:

- *CORA* $\widetilde{\bowtie}_{Publication.Title}$ *CORA*, a self join on only the publication's *Title* subterm;
- *CORA* $\widetilde{\bowtie}_{Publication.Venue}$ *CORA*, a self join on only the publication's *Venue* subterm;
- *CORA* $\widetilde{\bowtie}_{Publication.Coauthors}$ *CORA*, a self join on only the publication's *Coauthors* subterm, which is in turn a set of *Coauthor* subterms;
- *CORA* $\widetilde{\bowtie}_{Publication}$ *CORA*, a self join on the entire *Publication* term.

For each join, to keep results comparable, we choose the p-spectrum(2) kernel for strings and accept the default kernels for all other types. We do not optimise the default kernel for basic terms by choosing weighting modifiers that, for example, might be used to encode the intuition that a year of publication is less discriminating than the title of a publication when aggregated into an overall kernel on publications.

[1] The specific CORA data set used is an aggregation of all three CORA-REFS citation matching data sets (*fahl-labeled*, *kibl-labeled*, and *utgo-labeled*). The raw CORA-REF files have numerous XML mark-up errors which we have manually corrected to enabled parsing.

[2] *Year* is string rather than a numeric type due to non-numeric characters in the data. Also, *Venue* is constructed as a concatenation of venue-related fields; CORA has no venue field.

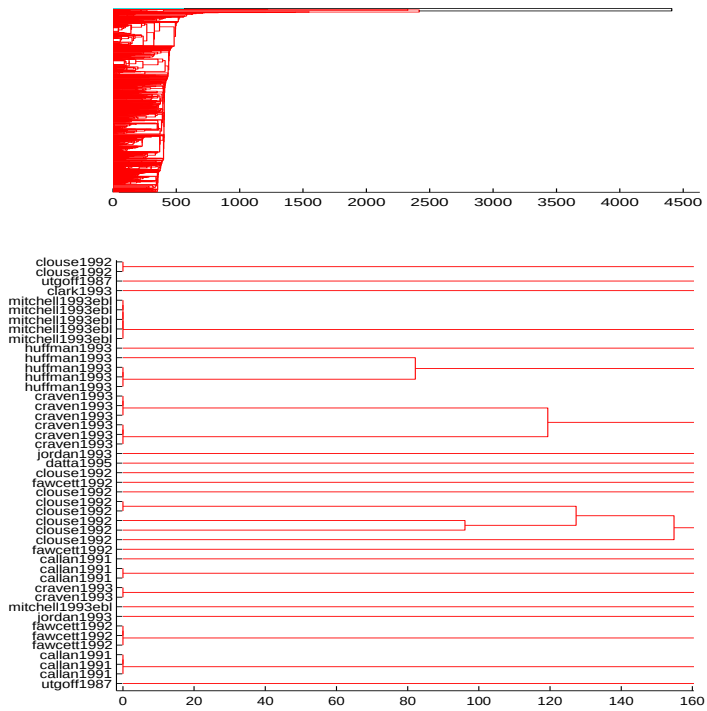

Fig. 5: Dendrogram (above), showing clusterings at successive thresholds for a proximity-joins on the CORA publication type, and (below) a close-up showing labelled ground truths.

For each of these four joins we constructed a dendrogram such as Fig.5 and calculated the corresponding precision-recall chart in Fig.6. Note that the trivial reflexive pairs, i.e. cluster sizes of 1, are ignored in the plots as they convey no useful information here and so lines are not interpolated to the top left of the chart (recall=0, precision=1). As would intuitively be expected, joins on *Publication.Title* is generally a better discriminator of publications than *Publication.Coauthors* and *Publication.Venue.* However, the default kernel for basic terms clearly effectively aggregates the information contained the subterms of *Publication* to outperform any single one of the three subterms taken in isolation. The only exception being *Publication.Title*, which sometimes outperforms its parent *Publication* above recall values greater than 0.9.

6 Related Work

Our work, to the best of our knowledge, is a unique combination of the relational model with a higher-order representation and distance-based methods. Thus we now describe our work with respect to related work in three related fields: relational learning, knowledge representation, and distances for structured data.

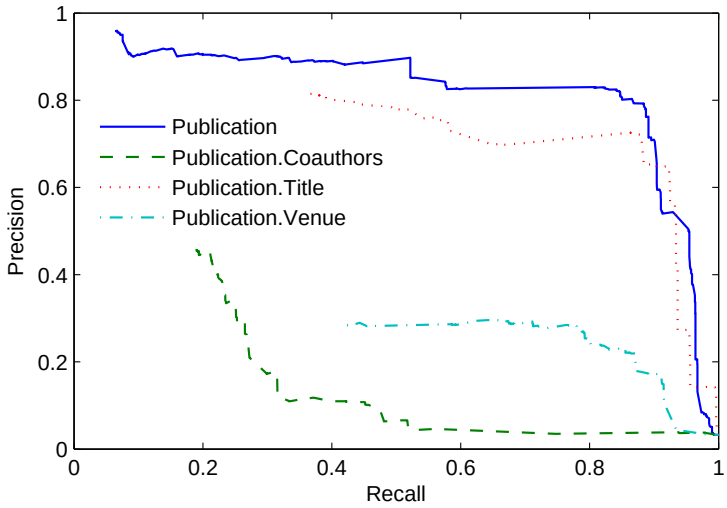

Fig. 6: Precision and recall for various decompositions of CORA publication type.

In database literature and more recently, particularly since the advent of the Web, within the relational learning literature, there has been considerable interest in the data integration aspects of database deduplication and record linkage [9, 10]. However, in addition to dealing with heterogeneous data structures, our work adopts an *individuals as terms* representation so that both type and structure of data is not obfuscated by traditional relational representations. Therefore our approach has the advantage of simplifying data modelling and the application of approximate matching techniques. Despite this, it should be noted that we propose the higher-order relational representation solely as an approach for data integration tasks, not as a replacement for general purpose relational databases. Our present implementation certainly has none of the optimisations of a modern relational database. Ultimately though, a higher-order view could be layered on top of a traditional relational database system, efficiently combining the two approaches, so that higher-order queries are automatically translated into and executed as equivalent relational queries.

Our goal of integrating and querying heterogeneous data is also a goal shared by the Semantic Web community [11]. The fundamental data model of the Semantic Web is the directed labelled graph, represented as RDF triples, which may be queried using the SPARQL query language [12]. Data structures such as lists, sets, multisets, trees and graphs are readily supported through RDF Schema and the OWL ontology language [13] and as such have similar representational advantages to basic terms as compared to the relational model. SPARQL queries can be used to retrieve a subgraph describing an individual that is analogous to a representation of that individual as a basic term. Conversely, it is straight forward to transform the same subgraph into a basic term in order to apply our own approach to RDF data. For RDF data integration, or *smushing* as

it is informally known, the emphasis in the Semantic Web languages to-date has been on exact matching, using inverse-functional properties such as email addresses, homepage URLs or entity URIs. This is an obvious shortcoming in the presence of noisy data or representational variations between data from different sources. To address the consequent data integration problem, work has been done in the area of ontology matching, including work on measuring proximity between ontologies [14]. Our approximate matching work differs from this explicit semantic integration approach in that we rely primarily on the implicit semantics of the type structure and data instances themselves. This is an advantage in cases where detailed ontological information is not available but potentially a disadvantage in other cases because background knowledge encoded in an ontology is not exploited in our approximate joins. The incorporation of background knowledge into our approximate joins is an area for future work.

Turning now to related work on distances, we first note that kernels and distances used in this paper are not of themselves a contribution of our work. Also, the choice of the default kernel for basic terms is not a specific requirement for the approximate relational join; any distance for basic terms would be suitable. Prior work on distances for logical terms includes distances between Herbrand interpretations [15] and between first-order terms (including structures and lists) [16–18]. None of these directly apply to basic terms and while it may be possible to apply distances on first-order terms to our first-order representation of basic terms, the semantics of basic abstractions would be lost as a result. Most closely related to our work, are various similarity-based methods that have been upgraded to handle structured data [4, 19, 20]. Contrasting approaches apply probabilistic models to take account of dependencies between resolution decisions [21, 22]. Most recently, a family of pseudo-distances over the set of objects in a knowledge base has been introduced although not specifically for basic terms [23].

7 Conclusion

In this paper we have combined two contrasting knowledge representational approaches, the relational model and basic terms, into a single coherent formalism that is well suited to the integration of heterogeneous data. This, in conjuction with the default kernel for basic terms has been shown to have potential for data integration and to be worthy of further investigation.

References

1. Codd, E.F.: The Relational Model for Database Management, Version 2. Addison Wesley (1990)
2. Date, C.J.: An Introduction to Database Systems. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1991)
3. Lloyd, J.W.: Logic and Learning. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2003)
4. Gaertner, T., Lloyd, J.W., Flach, P.A.: Kernels and distances for structured data. Mach. Learn. **57**(3) (December 2004) 205–232
5. Church, A.: A formulation of the simple theory of types. Journal of Symbolic Logic **5**(2) (June 1940) 56–68

6. Shawe-Taylor, J., Cristianini, N.: Kernel Methods for Pattern Analysis. Cambridge University Press (June 2004)
7. Gyftodimos, E., Flach, P.A.: Combining bayesian networks with higher-order data representations. In: Proceedings of the 6th International Symposium on Intelligent Data Analysis (IDA'06), Springer-Verlag (September 2005) 145–157
8. Culotta, A., McCallum, A.: Joint deduplication of multiple record types in relational data. In: CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management, New York, NY, USA, ACM (2005) 257–258
9. Lawrence, S., Bollacker, K., Giles, C.L.: Autonomous citation matching. In: Proceedings of the 3rd International Conference on Autonomous Agents, New York, NY, ACM Press (May 1999) 392–393
10. Newman, M.E.J.: The structure of scientific collaboration networks. Proc. Natl. Acad. Sci. USA 98 (2001) 404–409
11. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American (May 2001)
12. Prud'hommeaux, E., Seabourne, A.: SPARQL Query Language for RDF. W3C. W3C Working Draft 19 April 2005 edn. (April 2005)
13. McGuinness, D.L., van Harmelen, F.: OWL Web Ontology Language overview (2 2004)
14. Maedche, A., Staab, S.: Measuring similarity between ontologies. In: EKAW '02: Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web, London, UK, Springer-Verlag (2002) 251–263
15. Nienhuys-Cheng, S.H.: Distance between herbrand interpretations: A measure for approximations to a target concept. [24] 213–226
16. Sebag, M.: Distance induction in first order logic. [24] 264–272
17. Bohnebeck, U., Horváth, T., Wrobel, S.: Term comparisons in first-order similarity measures. In Page, D., ed.: ILP. Volume 1446 of Lecture Notes in Computer Science., Springer (1998) 65–79
18. Kirsten, M., Wrobel, S.: Extending k-means clustering to first-order representations. In Cussens, J., Frisch, A.M., eds.: ILP. Volume 1866 of Lecture Notes in Computer Science., Springer (2000) 112–129
19. Bhattacharya, I., Getoor, L.: Relational clustering for multi-type entity resolution. In: MRDM '05: Proceedings of the 4th international workshop on Multi-relational mining, New York, NY, USA, ACM Press (2005) 3–12
20. Woznica, A., Kalousis, A., Kalousis, M.H.A., Hilario, M.: Kernels over relational algebra structures. In: Proceedings of the 9th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining (PAKDD-2005). (2005) 588–598
21. Domingos, P., Domingos, P.: Multi-relational record linkage. In Dzeroski, S., Blockeel, H., eds.: Proceedings of the 2004 ACM SIGKDD Workshop on Multi-Relational Data Mining. (August 2004) 31–48
22. Bhattacharya, I., Getoor, L.: A latent Dirichlet model for unsupervised entity resolution. In: 6th SIAM Conference on Data Mining (SDM-2006), Bethesda, MD (2006)
23. d'Amato, C., Fanizzi, N., Esposito, F.: Induction of optimal semantic semi-distances for clausal knowledge bases. In Blockeel, H., Ramon, J., Shavlik, J.W., Tadepalli, P., eds.: ILP. Volume 4894 of Lecture Notes in Computer Science., Springer (2007) 29–38
24. Lavrac, N., Dzeroski, S., eds.: Inductive Logic Programming, 7th International Workshop, ILP-97, Prague, Czech Republic, September 17-20, 1997, Proceedings. In Lavrac, N., Dzeroski, S., eds.: ILP. Volume 1297 of Lecture Notes in Computer Science., Springer (1997)